A Lightweight Web Component Toolbox for Database-Driven Web Applications

Andreas Schmidt*‡ and Tobias Münch†§
* University of Applied Sciences
Karlsruhe, Germany
Email: andreas.schmidt@h-ka.de
‡ Karlsruhe Institute of Technology
Karlsruhe, Germany
Email: andreas.schmidt@kit.edu
† Münch Ges. für IT Solutions mbH, Germany
Email: to.muench@muench-its.de
§ Chemnitz University of Technology, Chemnitz, Germany

***Abstract*—Creating, content editing and interacting with relational databases in web applications has traditionally required developer knowledge of languages such as JavaScript or PHP. We present a lightweight, framework-independent toolbox for database-driven web applications. It provides intuitive database visualization, querying, and editing directly in a web browser—requiring only HTML knowledge. Unlike traditional frameworks like Angular or React, our web components simplify database interaction using a thin REST-based access layer. This work extends prior research by detailing the architecture, integration challenges, and considerations of extensibility.**

***Keywords*-Web Component; Relational-Database; Interface; Prototyping**

I. Introduction

This paper is an extended version of a conference paper [1], published in 2024 at the Sixteenth International Conference on Advances in Databases, Knowledge, and Data (DBKDA-2024) conference in Athens/Greece. In this extended paper, we examine the components we developed and describe their interactions in greater detail. We have also added an example scenario showing the components' use in a simple real-world scenario. In a detailed discussion, we also address current work in the area of security and authentication, as well as the expansion of the functionality of our components. We have also continued to work on linking the components to each other and to the other elements within the website. This means that the parameters of the components can now be read from the values of other components or HTML elements and monitored for changes. We have implemented a similar mechanism for the implementation of parameterized SQL statements.

With digitalization, there is increasing demand for user-friendly database interaction tools that support collaboration [2]. Traditionally, developers use frameworks such as Angular or React to create dynamic, database-driven web interfaces [3]. However, they also introduce noteworthy complexity, requiring developers to learn and maintain codebases, often leading to vendor lock-in [4], [5]. Non-developers can not cope with this complexity. The need for a lightweight and straightforward approach outranks the benefits of full-scale frameworks for scientific applications and internal business tools. For example, Microsoft Access has enabled user-driven database-driven development with minimal programming effort [6]. However, its desktop-based nature limits its accessibility and collaboration capabilities in modern web environments [6]. Additionally, in many companies, there is a shortage of suitable IT personnel and limited financial resources for IT services [7], [8]. The shortage of software engineers in particular poses a major challenge for companies [8], as they are needed to create and maintain large, complex software solutions. Reacting quickly to changing markets and optimizing processes are important criteria for remaining competitive. One way of overcoming these challenges is to expand the circle of potential developers. This is the path taken by *no-code* or *low-code* applications. One problem with this approach, however, is that the approaches are often proprietary and you get into a vendor lock-in, and the approaches are often difficult to maintain in a broader environment [9].

One way to get around this vendor lock-in is to use standards. In the last years, web components have emerged as an alternative for building reusable, framework-independent UI elements [10]. The World Wide Web Consortium (W3C) defines these components across web browsers [11]. These components enable developers to create custom HTML elements encapsulating functionality, style, and behaviour [10], [11]. Thus, they can be modular and easily integrated [11]. Compared to the development of database applications based on traditional programming languages, the development or integration of database applications using HTML code is less complex and therefore allows even business experts with basic HTML knowledge to implement or adapt an application according to their requirements and can therefore significantly reduce the workload of a company's IT-department.

As a consequence, we introduce a set of loosely coupled web components designed for database interaction functionality to web applications. Our approach is similar to adaptive Linked Data-driven Web components [12]. These components provide the following database operations:

- Table display and navigation (browsing relational database content)
- Dataset editing (modifying individual records)
- Query execution (dynamically running predefined SQL statements)
- Selection filtering (interactive searching and data retrieval)

These components communicate with databases via a REST-based access layer. Unlike traditional frameworks, this approach provides a lightweight, low-code alternative that enables non-experts to build database-driven web applications.

Unlike other low-code applications, which are often self-contained applications, we rely entirely on the W3C composite standard *Web Components* [13] and thus enable our components not only to create new applications, but also to be easily integrated into existing applications.

We structured the paper as follows: First, in Section II, we give an overview over related work. After that, we provide an overview of the composite standard *Web Components* in Section III. Then, we outline the system architecture in Section IV. Then, Section V presents the implemented components. The coupling of components with each other and with the other elements of the website is described in Chapter VI. Section VII demonstrates a real-world example application, while Section VIII discusses key challenges and limitations. Finally, Section IX concludes with future directions and planned enhancements.

II. Related Work

A. Database Access Through the Web

The classic approach to bringing database content to the web is via server-side programming such as PHP, Python or node.js (server-side JavaScript). To avoid having to constantly reinvent the wheel of web-based programming, frameworks based on these languages such as django (Python), symfony (PHP) or express (JavaScript) have emerged, most of which implement some form of the Model View Controller (MVC) paradigm and use an object relational mapping framework to access the database. In the field of database administration, phpMyAdmin [14] and derivatives such as pgMyAdmin are popular tools. Our approach differs fundamentally from this classic approach. Neither is the rendering done on the server side, nor is an imperative programming model used to create the content.

B. Java Applets

Java applets [15] took a different approach. These were typically small programs written in Java that were loaded from the server and ran in the user's browser in a protected environment (sandbox). A sophisticated access API for relational databases was also available through JDBC [16]. The integration of the applets into the web pages is done with the element `embed` or `object` and has a number of similarities with the web components approach we use. Both approaches run on the client within the browser and there are a number of predefined methods that must be implemented to integrate the application into the page. The integration is declarative in both cases and parameters can be passed to the program. Both approaches also allow the programmatic access to the DOM tree of the embedding website. From the mid-2010s, however, support for the applets was gradually discontinued by the browser manufacturers.

C. Declarative Web

At the ACM Web Conference 2023, Steven Pemberton delivered a presentation titled "The One Hundred Year Web," highlighting the escalating complexity and consequent formidable challenges in implementation [17]. Pemberton critiques the departure from the declarative path of the web with HTML5, referring to it as a "Cowpath" that is incongruent with the original principles [17]. He expresses hope that the damage incurred thus far can be rectified through the collaborative efforts of the web community, aiming to ensure the long-term backward compatibility of the web [17]. In our work, we want to follow exactly this approach and show that the otherwise imperative integration of database content can also be done declaratively.

Michael Hanus has introduced a concept that combines declarative programming with the use of a CGI program, which generates a web application based on the database [18]. Therefore, Hanus proposed an interface that integrates both functional and logical aspects derived from his Curry approach [18]. He posits that this conceptual framework is transferrable to a client-side context through the generation of JavaScript [18]. The primary distinction to our work lies in the rendering location of the HTML document. In the presented method, the HTML document is rendered on the server-side and transmitted to the client. On the other hand, in the client-side approach, the data is loaded from the server and rendered directly on the client-side.

D. Low Code Development

Low-code development platforms have the potential to significantly change the tasks of software engineering and help developers in companies create applications themselves without having to delve too deeply into coding, which significantly expands the pool of potential developers. Typically, low-code development platforms (LCDPs) provide a visual interface for application development that is realized through model-driven design and declarative programming [19], [20]. In addition to platform-specific platforms, there are also approaches that generate web applications as the target platform, such as the low-code platform Xelence from Sagitec Software [21].

The low-code initiative has given rise to a number of application developers, including Caspio [22], Budibase [23], webflow [24], and Bubble.io [25], which enable web-based applications to be developed in the form of single-page applications with little or no programming effort. However, these are standalone systems that are difficult or impossible to integrate into existing web applications. In contrast to these approaches, our low-code web components offer this functionality with ease of development, interoperability, extensibility, and maintainability.

III. Web Components

Web Components are custom, encapsulated, and reusable elements designed for integration into web pages or applications. They are processed and executed within contemporary web browsers. As of today, these APIs are integral components of

the Web Hypertext Application Technology Working Group (WHATWG) standard [26].

A *custom element* is a JavaScript class which can define custom HTML elements [26]. This element has to be inherited from the `HTMLElement` class [26].

The `HTMLElement` serves as the foundational class for every element present on a web page [26]. While it is possible to derive from a specific class like `HTMLParagraphElement`, such an approach is not fully supported by all browsers.

A Custom Element follows a specific lifecycle when invoked. Initially, the `constructor` is executed, establishing the initial configurations. Upon inclusion in the DOM, the `connectedCallback` method is triggered. Subsequently, the component is prepared and capable of both receiving and emitting events. If a property is changed, the `attributeChangedCallback` method is called, but only if the attribute is defined in the static property `observedAttributes`. Finally, when the element is removed from the DOM, the `disconnectedCallback` method is employed to internally reset the component [26].

Custom elements are registered by calling the method `define(tagName,class)` of class `customElements`. The method expects the tag name as the first parameter, so that it can be used in the markup, e.g. with `db-table`. The name of the implemented class is specified as the second parameter [26].

IV. Architecture of the Database web components

While the web components run in the browser, they have to communicate with a database server. The developed web components don't communicate directly with the database, but through a thin Representational State Transfer (REST)-based access layer (see Fig. 1, middle). This service maps a logical database identifier to a specific database on the server side. We use the PHP-CRUD-API library [27] by Maurits van der Schee as the core for this purpose and have extended it with additional functionality. PHP-CRUD-API provides a REST-based CRUD interface for accessing relational databases, i.e., records can be created, read, updated, and deleted. We implemented the necessary extension modules ourselves, such as access to the database metadata and a module for executing SQL statements, and integrated them into the PHP-CRUD-API as *customControllers* [27]. In order to be able to handle multiple databases per endpoint, we have also implemented the wrapper module `rdbms.php`, which provides additional meta information about the databases available at the endpoint and initializes the PHP-CRUD-API module with the specific, selected database.

Specifically, the service provides the following functionality:

Database scheduler:
: The REST API can manage multiple databases running on any computer. For this purpose, the new entry point *rdbms.php* has been implemented, which takes over the management of the databases. This also includes providing meta information about which databases are available via this endpoint. Once a specific database has been specified, the *rdbms.php* module forwards the request to the PHP-CRUD-API module for processing.

Access Control:
: In the configuration for accessing the databases, it can be specified which tables are accessible. In addition to this coarse-grained access control, the PHP-CRUD-API library offers further mechanisms such as authentication via API key, JWT token, or username/password. Currently, we are implementing authentication based on Keycloak [28] via JWT token forwarding.

CRUD-Functionality:
: This functionality is completely coverd by the PHP-CRUD-API library and includes the (C)reation, (R)eading, (U)pdating, and (D)eletion of datasets in the database.

Metadata Module:
: For the purpose of constructing forms for creating and modifying data records, as well as for resolving foreign key relationships, we require information about the structure of the tables and their constraints. We have implemented this functionality as a custom controller of the PHP-CRUD-API module. This has the advantage that the authentication mechanisms used by PHP-CRUD-API can also be used for this module.

SQL Module:
: The SQL module was also implemented as a custom controller of the PHP-CRUD-API library for the same reasons as the metadata module. It allows the execution of parameterized SQL select statements.

V. Components

As part of our research work, we have developed a series of web components for the declarative integration of database functionality into HTML pages. Concrete, the following components were realized:

db-connection: This component establishes the connection between the components and the RESTful backend service. It acts as an intermediary between the other components and the RESTful service. Additionally, it is also responsible for authentication based on Keycloak via JWT token forwarding.

db-table: This component represents a database table. The functionality ranges from the simple display of data records to a wide variety of interaction options like sorting, further filtering an so on.

db-row: Component for representing a single data set (row in a table). The functionality of this component ranges from simple, non interactive visualization of the data set to the creation and editing of data sets using predefined or freely definable forms, and the use as a controller in a model-view-controller (MVC) szenario.

db-field: This component represents a single attribute of a data set. `db-field` components are used in conjunction

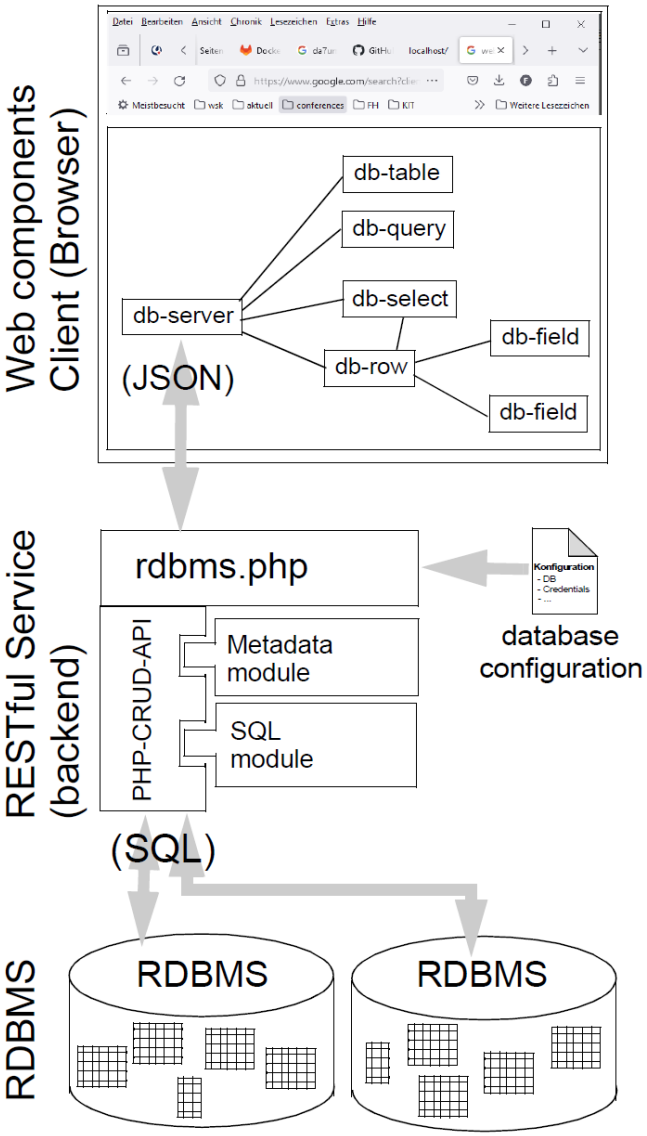


Figure 1. Architecture of our database web components.

with the `db-row` component when arbitrary layouts are to be realized for the visualization of a data set.

db-select: Analogous to the HTML select element, in which a value can be selected from a list of predefined values. In the case of the `db-select` component, the displayed values come from the results of an SQL query or the selection of certain columns from a table.

db-query: Component that displays the results of an arbitrary SQL query. It is also possible to browse through the results and resort them.

In the following we will present these components in detail. The data in the example screenshots shown comes from the *Mondial* database [29]. The PHP-CRUD-API library we use does not support composite keys for write operations. For this reason, we modified the *Mondial* database schema and added artificial keys and corresponding foreign keys.

A. Connection-component

The connection-component is a non-visible component in a page. It is responsible for the mapping to a concrete database on server-side. The left side of Fig. 2 gives an example, how the web component is integrated inside a HTML page. The `db-connection` component communicates with a RESTful service, which is specified by the parameter `url`. The further parameter `database` specifies a logical database name, which is mapped on server side (Fig. 2, middle) to a specific database (right side of Fig. 2). Note that the database can run on any computer and not necessarily on the computer with the REST-API endpoint. Table I shows all possible attributes of the component.

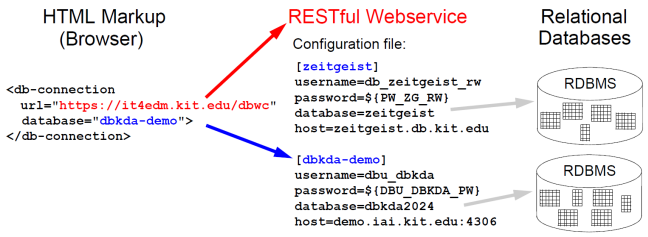


Figure 2. Database mapping (from [30]).

TABLE I
`db-connection` ATTRIBUTES

Attribute	Description	Mandatory
database	Logical database name. This name is mapped to a concrete database on the backend side.	yes
url	URL of the REST-API endpoint Default: URL, from where the web-components are loaded	no

B. Table-component

The `db-table` component is responsible for displaying the content of a database table or a part of it. Fig. 3 shows in the upper part the definition of a `db-table` component. The table to be shown is "country", of which the three columns `Name`, `Capital`, and `Population` are to be displayed (attribute `attribute-list`). The `actions` attribute specifies the possible interaction options. In this specific case, page-by-page scrolling (`paging`) is enabled with a page size of 10 datasets (parameter `pagesize`), the datasets can be sorted in ascending and descending order according to the column values (`sort`), and additional filtering can be carried out at column level (`filter`). Inline-edit is also activated (`inline-edit`).

In the lower part of Fig. 3 the visual appearance of the component inside the browser, according to the previously described specification is shown. Paging (1), sorting (2), filtering (3) and inline-edit (4) as specified in the markup are activated. The full list of possible attributes of the component are shown in Table II. The `refresh-rate` attribute, for example, is responsible for ensuring that the current database content is always displayed by accessing the database table

again every n seconds and rereading and displaying the actual values. In addition to the parameters from Table II, the appearance of the component can also be adapted to your own requirements using cascading stylesheets (CSS).

```
<db-table table="country"
          actions="paging,sort,filter,inline"
          pagesize="10"
          attribute-list="Name,Capital,Population">
</db-table>
```

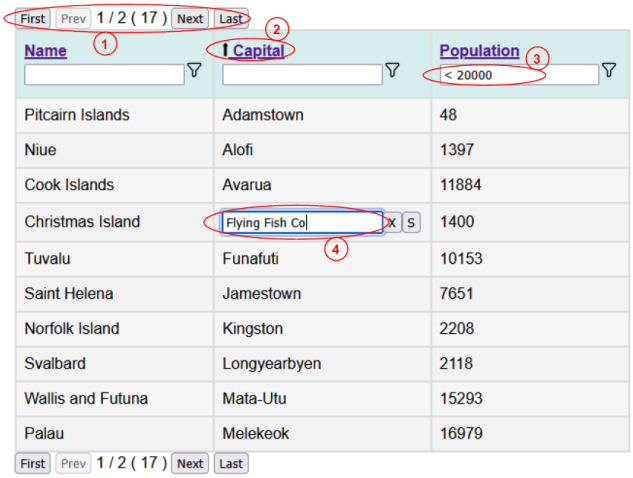


Figure 3. Specification and visual appearance of the web component `db-table`, showing different interaction elements like page-wise scrolling (1), sorting (2), filtering (3), and inline-editing (4).

TABLE II
`db-table` ATTRIBUTES

Attribute	Description	Mandatory
table	Name of the table	yes
filter	Mandatory filter condition, that all datasets must fulfill	no
pagesize	Maximum number of datasets on a page	no
page	Page to display	no
order	Sort order (column name)	no
direction	Sort direction (`asc`, `desc`)	no
connection	Id of a `db-connection` web-component. If the attribute is not set, the default server component is chosen	no
attribute-list	Comma separated list of attributes to display (default: all)	no
actions	List of possible values: sort, filter, paging, inline, edit, delete	no
refresh-rate	Time in seconds after which the table data is reloaded from the database	no

C. Selection-component

The `db-select` web component presents a selection box, from which values can be selected and searched via a prefix or infix search. The values are specified by an SQL-select statement. The SQL-statement can either be specified directly by the `sql`-attribute, or it is specified using the attributes `value`, `text`, `table` and (optional) `filter`. On the left side of Fig. 4, the visual appearance with prefix-search is shown, while on the right hand side, the markup, defining the web-component on the left, is shown. The attribute `value` represents the table-column, which values are passed to the HTML-form on submit, while the values of the column, specified by the attribute `text` are displayed by the element and are used for the prefix/infix-search. The attribute `name` specifies the name of the `db-select` element, and thus the name of the attribute under which the selected value is transferred to the form. Table III provides an overview of all possible attributes and their meaning.


```
<db-select
    table="country"
    value="Code"
    text="Name"
    name="Country_Fk">
</db-select>
```

Figure 4. web component `db-select`

TABLE III
`db-select` ATTRIBUTES

Attribute	Description	Mandatory
connection-id	Id of a db-connection web-component	no
prefix-search	The term entered in the search field is considered as a prefix (Default: true)	no
name	Name of the element. The value of the selected entry (attribute `value` in Variant I, first column of SQL-statement in Variant II) is passed on to the form under this name.	no
Variant I		
table	Name of the table	yes
value	Name of the attribute used for the value-attribute of the option element (returned by the form).	yes
text	Name of the attribute used for the text-node of the option element (value used for search).	yes
filter	Mandatory condition the datasets must fulfill.	no
Variant II		
sql	SQL-Statement with one or two columns. The statement can have one or two columns in the select-clause. If only one column is given, the value of this column is used for the value and the text.	yes

D. Row-component

The component represents a single dataset. It allows you to create a new record or edit an existing one. The component provides its own HTML form for this purpose. To do this, the component uses the functionality provided by the metadata module to determine the structure and constraints of the table. This concerns information about the names and data types of the fields, `not null` constraint and information about primary and foreign key relationships. With the help of this information, foreign key relationships, for example, are resolved by displaying not the foreign key value but the referenced data

record. This feature is shown on the left side of Fig. 5, where the foreign key attributes `Capital_fk` (label: Capital), which reference the capital city of a country, is not displayed, but a selection box showing the current value and simply giving the option to change it using the `db-select` web-component described previously. If a renaming of the attribute identifiers of the database table is desired (as here, `Capital` instead of `Capital_fk`), this can be accomplished with CSS rules.

The right side of Fig. 5 shows, how the markup of the web-component, shown on the left, looks like. The parameter `key` expects the value of the primary key of the dataset. The name of the primary key attribute does not have to be specified as it is determined from the metadata of the table. If the attribute `key` is omitted, the component provides a form within which a new data record can be created.

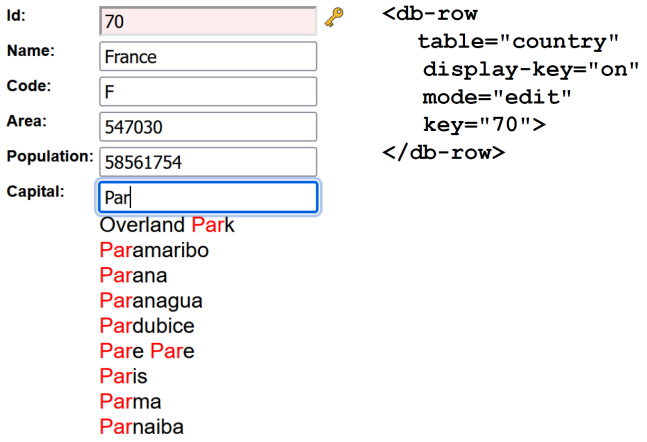


Figure 5. Web-component `db-row` in edit-mode using the predefined layout. Foreign keys (i.e., Capital) are represented by `db-select`-components, allowing to search and select a concrete dataset.

Another use of the `db-row` component is as a controller in a model-view-controller (MVC) setup (attribute `controller="true"`). In this case, the component is invisible and reads the parameters either from the GET parameters of the current URL or from a JSON string that has been passed to the `data`-attribute. Depending on whether the primary key value has been specified, an SQL update or insert operation is performed. Subsequently, the page specified by the `target-url` is loaded (or `error-url` in case of an error).

Table IV lists all possible attributes of the web component `db-row`. The `form` attribute, for example, is used to create your own HTML form layout and link it to the component, so that this form is used instead of the internal one. If the parameter `is-editable` is set to `false`, a predefined, read-only representation of the dataset is displayed. If this does not meet layout requirements, the `visible` attribute can be set to `false` and the layout can be defined using `db-field` components (see below).

E. Field-Component

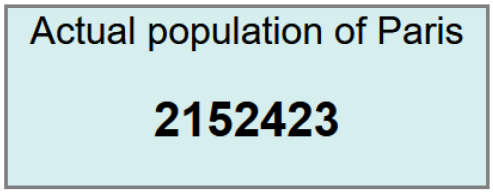
The `db-field` component handles exactly the value of one column of a dataset. It gets its value from a `db-row` component and returns the value within a `<span>` element. If only one `db-row` is specified on the HTML-page, this is used automatically, otherwise the desired `db-row` must be specified with the parameter `dataset`. Fig. 6 gives a minimal example, displaying the name and population of the city with the primary key value of 2037 (Paris). In the upper part, you see the result in the browser, at the bottom the corresponding declaration of the elements `db-row` and two `db-field` elements in the HTML page. Note that in the `db-row` component, the attribute `refresh-rate` is set to the value of 10. This ensures that the dataset is reloaded every 10 seconds so that the actual number of inhabitants is always displayed.

TABLE IV
`db-row` ATTRIBUTES

Attribute	Description	Mandatory
table	Name of the table	yes
connection-id	Id of a db-connection web-component	no
visible	Specifies if the component should display the dataset on the page. If the component is used together with `db-field` components you typically set this parameter "off" (default: on)	no
is-editable	Specifies if the dataset should be editable. Possible values: yes/no(default: yes)	no
display-key	Flag, if the database key should be displayed or not	no
attribute-list	Comma separated list of attributes to be shown (default: all attributes)	no
key	The value of the primary key of a dataset in this table.	no
form	the id of a form-element, the `db-row` component should work with. Additionally a mapping between the form fields and the table columns can be specified using the *mapping* attribute (see below).	no
mapping	Mapping between form field name and table column name.	no
controller	Enable "controller-mode". If the parameter is set to "true", the parameter `action` must also be set (`store-from-get-request`, `store-from-data`).	no
action	See description above (controller-mode).	no
target-url, error-url	Page to load after dataset is written (controller-mode).	no
refresh-rate	Duration (in seconds) until the dataset is read again	no

F. Query-Component

The visual representation of the `db-query` component is similar to that of the `db-table` component in Fig. 3. The main difference is that no `table` parameter is specified, but an arbitrary SQL select-statement. Fig. 7 shows an example in which the SQL statement determines the number of borders and overall border length for all countries having more than 5 neighbors, sorted by decreasing number of neighbors and decreasing border length. Table V provides a list of all possible attributes.




```
<db-row table="city" visible="off" key="2037"
        refresh-rate="10">
</db-row>
<div class="large-box">
    Actual population of <db-field attribute="Name">
                         </db-field>
    <p/>
    <db-field class="bold-huge" attribute="Population">
    </db-field>
</div>
```

Figure 6. Two `db-field` components with an invisible `db-row` component

first prev [1 - 10] (32) next last

name	# neighbors	border_length
China	16	22143
Russia	14	19913
Brazil	10	14691
Zaire	9	10730
Sudan	9	7687
Germany	9	3621
Tanzania	8	3861
France	8	2892
Turkey	8	2627
Austria	8	2558

first prev [1 - 10] (32) next last

```
<db-query
   sql="select c.name, count(*) as &quot;# neighbors&quot;,
               round(sum(length)) as border_length
          from country c
          left join borders b
            on b.Country1_fk=c.Id or b.Country2_fk=c.Id
         group by c.Id, c.Name
        having count(*) > 5
         order by 2 desc, 3 desc"
   pagesize="10"
   actions="paging,sort">
</db-query>
```

Figure 7. Web-component `db-query` executing a complex SQL statement. The result datasets can be scrolled and ordered by the different columns of the result.

One of the most important enhancements since the version of our components described in [1] is the support of parameterizable SQL-statements. Instead of values, it is possible to define placeholders in the SQL-statement in the form of question marks (?), which get their value from other HTML elements of the website. This can be, for example, the value of an input field, or the currently selected value of an HTML-selectbox or the *db-select* (see Section V-C) component implemented by us. Fig. 8 shows this functionality, where the SQL-statement contains two parameters that take their values from the two input fields that specify the minimum and maximum height of the mountains to be displayed. The mapping between the input fields and the parameters in the SQL statement is realized by the attribute `params` of the component `db-query`. The values of the attribute specify the id of the HTML components and the property to be read. In the case of the HTML input element, this is the property `value`.

Height in between 500 and 1500 metres

first prev [1 - 10] (22) next last

name	height	mountains	type
Feldberg	1493	Schwarzwald	
Hekla	1491		volcano
Pinatubo	1486		volcano
La Soufriere	1467		volcano
Puy De Dome	1465	Cevennes	volcanic
Grosser Arber	1456	Bayrischer Wald	
Katla	1450		volcano
Morne Diablotins	1447		volcano
Grand Ballon	1424	Vosges	
Pelee	1397		volcano

```
<form>
  Height in between
  <input type="number" id="low-limit"  value="500"> and
  <input type="number" id="high-limit" value="1500">
 metres
</form>
<p>
<db-query sql="select name, height, mountains, type
                 from mountain
                where height >= ?
                  and height <= ?
                order by height desc"
          params="low-limit.value high-limit.value"
          pagesize="10">
</db-query>
```

Figure 8. Web-component `db-query` with a parameterized SQL-statement, reading the values from two input fields.

By specifying the two input elements from which the values for the parameterized SQL-statement are read, event handlers are also registered, which inform the `db-query` component about changes to the values for the input fields, so that the component executes the SQL-statement again when the values change.

TABLE V
`db-query` ATTRIBUTES

Attribute	Description	Mandatory
sql	SQL Statement to be executed	yes
pagesize	Maximum number of datasets on a page	no
page	Page to display	no
connection	Id of a `db-connection` web-component. If the attribute is not set, the default server component is chosen	no
refresh-rate	Duration (in seconds) until the query is resubmitted	no
actions	List of possible values: sort, paging	no

VI. PARAMETER BINDING

Analogous to the behavior of parameterized SQL statements, we have implemented a mechanism that allows the values of the parameters of our components to be read from other

components. This is achieved by a special syntax, in which instead of the value for an attribute, the id of the HTML component followed by the concrete property (dot-notation) is specified within double curly brackets. Fig. 9 demonstrates this using the example of the `pagesize` attribute of the `db-table` component. Instead of specifying the value as fixed, the `{{..}}` notation is used here to reference the property `value` of the select element with the ID `choose-pagesize` (`{{choose-pagesize.value}}`).

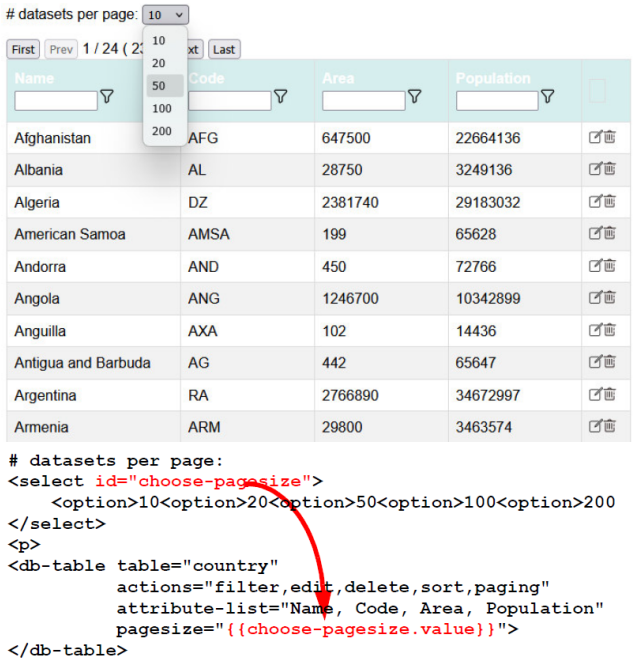


Figure 9. Parameter binding: Mapping attribute values to other HTML-elements.

VII. Example Application

In this section, the use of the components we have developed will be demonstrated using a small example application. The components `db-connection`, `db-table` and `db-row` will be used. The aim of the application is a simple task list in which tasks to be completed can be entered and managed with a priority and optional deadline. The application consists of two HTML pages, a view and a controller page. The view is shown in Fig. 10 and contains a HTML-form for entering a new task and below it a table with the tasks already created. Beside the form-element with the input fields for the new task, the page contains the web-components `db-connection` and `db-table`.

When the submit button ("add task") is pressed, the form data is send to the page `iaria-demo.controller.html` (see Fig. 11) via the GET-method and this page acts as a controller like in a MVC setup, storing the passed values and after that reload the calling page. The complete code of the controller file `iaria-demo.controller.html` is shown in Fig. 11. In line 2 and 3, our database web-components are loaded. Line 4 and 5 sets up the component `db-connection` to establish a connection to the database with the name `iaria-demo`. If the attribute `url` is omitted, it is expected that the endpoint resides at the same address, from where the components were loaded (see line 2). In lines 6 to 8, the component `db-row` is configured as a controller (`controller="true"`), reads the parameters passed from the previous page (the view with the form from Fig. 10) and inserts the dataset into the `todo_list` table. Since no `target-url` or `error-url` parameter are specified, the previous form page is reloaded after the insert operation is executed.

VIII. Discussion, Further Challenges

Our web components for database developers serve as a proof of concept, demonstrating that lightweight, framework-independent database interaction is feasible. However, we must work towards a low-code or no-code solution to transition from a developer-focused prototype to a solution for no-coders. A visual tool allowing users to connect components dynamically could improve accessibility - such as a split-screen UI with commands on the left and live interaction on the right.

Challenges such as validation, scalability, and security are critical for real-world usage. Performance testing is needed to ensure components handle large datasets and concurrent users efficiently, while usability testing will determine if non-developers can use the system effectively. Hosting components on a CDN and optimizing the backend REST API will enhance scalability. Security remains a primary concern, requiring role-based access control (RBAC) to restrict data access. Additionally, input sanitization is necessary to prevent SQL injection and XSS attacks.

To address these challenges, we will:

1) Develop a wizard-based UI for configuring components without coding.
2) Implement authentication and authorization mechanisms for secure user access on the World Wide Web.
3) Integrate components into third-party web applications for real-world test settings to evaluate arising problems.

Our database components could enhance scalability, security, and usability and serve as an alternative to traditional methods, effectively connecting developer-driven tools and no-code solutions.

IX. Conclusion and Outlook

We have implemented the first prototype of our web components and are actively working on expanding their functionality. One key improvement is evaluating all available database metadata directly within the components. This metadata is already visible in Fig. 5, where a selection box displays a dereferenced value (e.g., "Paris") instead of the foreign key for the capital.

Security remains a significant area for future work. Actually we allow the restriction to only acces certain tables inside a database and support authentication based on keycloak via JWT token forwarding. For future iterations we plan to implement a role-based access control (RBAC) meachanism. Using our web components in another web application with RBAC makes

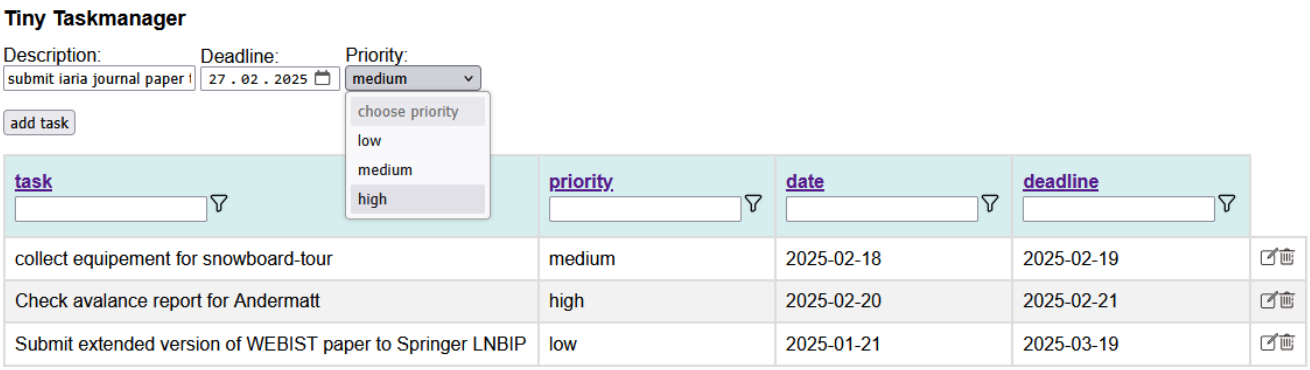


Figure 10. Example Application: View with form and `db-table` component.

```
1:  <!-- controller file: iaria-demo.controller.html -->
2:  <script type="module" src="http://localhost/dbwc/db.js">
3:  </script>
4:  <db-connection database="iaria-demo">
5:  </db-connection>
6:  <db-row controller="true" table="todo_list"
7:          action="store-from-get-request">
8:  </db-row>
```

Figure 11. `db-row` component, acting as a controller (complete code).

the integration possibilities interesting and the security aspects challenging.

Several functional enhancements are also in progress. The `db-select` component will be extended to support multiple selections, making handling $n : m$ relationships easier. The `db-table` component currently does not resolve foreign keys, but this can be worked around using `db-query` with SQL-join operations. In the future, native support for foreign key dereferencing in the `db-table` will be added as an optional feature.

For future work, we focus on four key areas:

1) **Security and Authentication** – Adding built-in authentication and authorization mechanisms.
2) **Scalability and Performance** – Optimizing data handling for large datasets and concurrent users.
3) **No-Code Accessibility** – Developing a visual configuration wizard for non-developers.
4) **Evaluation** - Evaluation of our framework in terms of the time required to develop a specific application. This should include a comparison with traditional software development approaches as well as with modern existing low-code solutions.

Future work will also involve real-world testing and integration into enterprise applications to gather feedback and further refine the components. Our components will enable business developers to create new business cases with software support without needing software engineers.

References

[1] A. Schmidt and T. Münch, "Web components for database developers", in *Proceedings of the Sixteenth International Conference on Advances in Databases, Knowledge, and Data Applications*, 2024, pp. 20–22.

[2] H. Chen, R. H. Chiang, and V. C. Storey, "Business intelligence and analytics: From big data to big impact", *MIS quarterly*, pp. 1165–1188, 2012.

[3] R. Vyas, "Comparative analysis on front-end frameworks for web applications", *International Journal for Research in Applied Science and Engineering Technology*, vol. 10, no. 7, pp. 298–307, 2022.

[4] J. Hassan, "The effects of architectural design decisions on framework adoption: A comparative evaluation of meta-frameworks in modern web development", Ph.D. dissertation, May 2024. DOI: 10.13140/RG.2.2.10552.97287.

[5] C. Shapiro, *Information rules: A strategic guide to the network economy*. Harvard Business School Press, 1999.

[6] J. Eckstein and B. R. Schultz, *Introductory relational database design for business, with Microsoft Access*. John Wiley & Sons, 2018.

[7] K. Almaree *et al.*, "The usefulness of cash budgets in micro, very small and small retail enterprises operating in the cape metropolis", *Expert Journal of Business and Management*, vol. 3, no. 1, 2015.

[8] M. Skare, M. d. l. M. de Obesso, and S. Ribeiro-Navarrete, "Digital transformation and european small and medium enterprises (smes): A comparative study using digital economy and society index data", *International journal of information management*, vol. 68, p. 102 594, 2023.

[9] K. Rokis and M. Kirikova, "Challenges of low-code/no-code software development: A literature review", in *International conference on business informatics research*, Springer, 2022, pp. 3–17.

[10] T. Münch, "Vanilla js-design and implementation of a progressive web application from scratch", in *International Conference on Web Engineering*, Springer, 2024, pp. 461–464.

[11] D. Glazkov and H. Ito, *Introduction to web components*, https://www.w3.org/TR/components-intro/, [Online; accessed 2025-05-27].

[12] A. Khalili, A. Loizou, and F. van Harmelen, "Adaptive linked data-driven web components: Building flexible and reusable semantic web interfaces: Building flexible and reusable semantic web interfaces", in *The Semantic Web. Latest Advances and New Domains: 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29–June 2, 2016, Proceedings 13*, Springer, 2016, pp. 677–692.

[13] "Web components - specifications", [Online; accessed 2025-05-27], 2015, [Online]. Available: https://www.webcomponents.org/specs.

[14] M. Delisle, *Mastering phpMyAdmin 3.1 for Effective MySQL Management*. Packt Publishing, 2009, ISBN: 1847197868.

[15] E. Boese, *An Introduction to Programming With Java Applets*. Jones and Bartlett Publishers, 2009.

[16] G. Reese, *Java Database Best Practices: Persistence Models and Techniques for Java Database Programming*. O'Reilly, 2009.

[17] S. Pemberton, "The one hundred year web", in *Companion Proceedings of the ACM Web Conference 2023*, ser. WWW '23 Companion, Austin, TX, USA: Association for Computing Machinery, 2023, pp. 642–647, ISBN: 9781450394192. DOI: 10.1145/3543873.3585578.

[18] M. Hanus, "Lightweight declarative server-side web programming", in *Practical Aspects of Declarative Languages: 23rd International Symposium, PADL 2021, Copenhagen, Denmark, January 18-19, 2021, Proceedings*, Copenhagen, Denmark: Springer-Verlag, 2021, pp. 107–123, ISBN: 978-3-030-67437-3. DOI: 10.1007/978-3-030-67438-0_7.

[19] E. Elshan, E. Dickhaut, and P. Ebel, "An investigation of why low code platforms provide answers and new challenges", in *Hawaii International Conference on System Sciences (HICSS)*, (Maui, Hawaii), Maui, Hawaii, 2023.

[20] N. Prinz, C. Rentrop, and M. Huber, "Low-code development platforms-a literature review.", in *AMCIS*, 2021.

[21] R. Arora, N. Ghosh, and T. Mondal, "Sagitec software studio (s3)-a low code application development platform", in *2020 International Conference on Industry 4.0 Technology (I4Tech)*, IEEE, 2020, pp. 13–17.

[22] Caspio, *Caspio: Low-Code Platform - Build Online Database Apps*, https://www.caspio.com/, (Accessed on 2025-05-23), 2024.

[23] Budibase, *Github: Budibase/budibase*, https://github.com/Budibase/budibase, (Accessed on 2025-05-23), 2024.

[24] Webflow, *Webflow: Create a custom website | Visual website builder*, https://webflow.com/, (Accessed on 2025-05-23), 2024.

[25] Bubble, *Bubble: The full-stack no-code app builder*, https://bubble.io/, (Accessed on 2025-05-23), 2024.

[26] *WHATWG. HTML Living Standard*, https://html.spec.whatwg.org/multipage/, [Online; accessed 2025-05-27].

[27] M. van der Schee, *PHP-CRUD-API*, https://github.com/mevdschee/php-crud-api, Last accessed 17.2.2025, 2025.

[28] S. Thorgersen and P. I. Silva, *Keycloak-identity and access management for modern applications: harness the power of Keycloak, OpenID Connect, and OAuth 2.0 protocols to secure applications*. Packt Publishing Ltd, 2021.

[29] W. May, "Information extraction and integration with FLORID", Last accessed 17.2.2025, 1999, [Online]. Available: http://dbis.informatik.uni-goettingen.de/Mondial.

[30] A. Schmidt and T. Münch, "Enable Business Users to Embed Dynamic Database Content in Existing Web-Based Systems Using Web Components and Generic Web Services", in *Proceedings of the 20th International Conference on Web Information Systems and Technologies - WEBIST*, 2024, pp. 296–306. DOI: 10.5220/0013000000003825.